In the Claims:

Please cancel claims 2, 6-8, 10-13, and 15-18. Please amend claims 1 and 3-5. Please add new claims 20-27. The claims are as follows:

1. (Currently amended) A method ~~of determining, in a computer environment, the equivalence, if any, of two algebraic expressions for use in compiler optimisation of source code and like computing tasks, said method~~ for performing compiler optimisation of source code during compilation of the source code in a computer environment by determining and utilizing the equivalence of two syntactically correct algebraic expressions comprised by the source code, said method comprising compiling the source code into object code, said compiling comprising determining the equivalence of the two algebraic expressions followed by eliminating a redundant algebraic expression of the two algebraic expressions determined to be equivalent, said determining the equivalence of the two algebraic expressions comprising the steps of:

(a) recasting said expressions into a form of one or more token pairs arranged sequentially in a string, each said token pair comprising an operator followed by an operand;

(b) reducing said strings in accordance with a set of predetermined simplifying rules;

(c) comparing the reduced strings by matching, to detect equivalence of the two algebraic expressions; ~~and~~

~~(c1) compiling said source code into object code, wherein said source code comprises said two algebraic expressions, and wherein said compiling comprises said recasting, said reducing, and said comparing,~~

wherein said determining the equivalence of the two algebraic expressions comprises prior to the recasting step (a) ~~is preceded by a preconditioning step comprising~~, in relation to said algebraic expressions, at least one of the following preconditioning sub-steps:

(da) deleting a space in the expression;

(db) removing a bracket in the expression by expanding a bracketed sub-expression;

(dc) inserting a unitary operator at the start of the expression;

(dd) recasting a power factor, being a variable being raised to a power in the expression, in an alternate form as one of:

(dda) the power factor being expressed as the variable multiplied by itself as many times as the power, if the power is a positive integer;

(ddb) the power factor being expressed as a reciprocal of the variable multiplied by itself as many times as an absolute value of the power, if the power is a negative integer;

(ddc) the power factor being replaced by an appropriate function which can compute the power factor, if the power is not an integer;

(de) recasting a constant in the expression in exponential format;

(df) substituting a "+" operator in the expression by "+1*", a "1" being in exponential format;

(dg) substituting a "-" operator in the expression by "-1*", a "1" being in exponential format; and

(dh) recasting a "division by a constant" in the expression as multiplication by a reciprocal of the constant;

wherein said reducing said strings in accordance with the set of predetermined simplifying rules in step (b) comprises:

(ba) arranging token pairs into subgroups;

(bb) arranging operand tokens in an arranged subgroup in order;

(bc) reducing the ordered operands by consolidating one or more constants and eliminating variables of opposite effect to form reduced subgroups; and

(bd) consolidating one or more multiple instances of similar subgroups, to produce a reduced string.

2. (Canceled)

3. (Currently amended) The method of claim 1, wherein said determining the equivalence of the two algebraic expressions comprises prior to the recasting step (a), in relation to said algebraic expressions, all of the preconditioning sub-steps ~~the simplifying rules in step (b) comprise:~~

~~(ba) arranging token pairs into subgroups;~~

~~(bb) arranging operand tokens in an arranged subgroup in order;~~

~~(bc) reducing the ordered operands by consolidating one or more constants and eliminating variables of opposite effect to form reduced subgroups; and~~

~~(bd) consolidating one or more multiple instances of similar subgroups, to produce a reduced string~~.

4. (Currently amended) The method of claim 1, wherein an algebraic expression whose

equivalence is to be determined contains an aliased variable, said ~~method~~ determining the equivalence of the two algebraic expressions further comprising the steps of:

arranging an ordered list of aliases of the variable, and substituting a first alias in the ordered list for all instances of the aliased variable in the expression, wherein said compiling comprises said arranging an ordered list of aliases of the variable and said substituting a first alias in the ordered list.

5. (Currently amended) The method of claim 1, wherein an algebraic expression whose equivalence is to be determined contains a function, said ~~method~~ determining the equivalence of the two algebraic expressions further comprising the steps of:

reducing function arguments using the set of predetermined simplifying rules; and replacing the function by a tagged string, said string designating a function name, parameter types, and arguments, wherein the tag distinguishes the function name from a variable, wherein said compiling comprises said reducing function arguments and said replacing the function by a tagged string.

6-19. (Canceled)

20. (New) A computer program product, comprising a computer readable storage medium having a computer readable program code embodied therein, said computer readable program code containing instructions that when executed by a processor of a computer system implement a method for performing compiler optimisation of source code during compilation of the source

code in a computer environment by determining and utilizing the equivalence of two syntactically correct algebraic expressions comprised by the source code, said method comprising compiling the source code into object code, said compiling comprising determining the equivalence of the two algebraic expressions followed by eliminating a redundant algebraic expression of the two algebraic expressions determined to be equivalent, said determining the equivalence of the two algebraic expressions comprising the steps of:

(a) recasting said expressions into a form of one or more token pairs arranged sequentially in a string, each said token pair comprising an operator followed by an operand;

(b) reducing said strings in accordance with a set of predetermined simplifying rules;

(c) comparing the reduced strings by matching, to detect equivalence of the two algebraic expressions;

wherein said determining the equivalence of the two algebraic expressions comprises prior to the recasting step (a), in relation to said algebraic expressions, at least one of the following preconditioning sub-steps:

(da) deleting a space in the expression;

(db) removing a bracket in the expression by expanding a bracketed sub-expression;

(dc) inserting a unitary operator at the start of the expression;

(dd) recasting a power factor, being a variable being raised to a power in the expression, in an alternate form as one of:

(dda) the power factor being expressed as the variable multiplied by itself as many times as the power, if the power is a positive integer;

(ddb) the power factor being expressed as a reciprocal of the variable multiplied by itself as many times as an absolute value of the power, if the power is a negative integer;

(ddc) the power factor being replaced by an appropriate function which can compute the power factor, if the power is not an integer;

(de) recasting a constant in the expression in exponential format;

(df) substituting a "+" operator in the expression by "+1*", a "1" being in exponential format;

(dg) substituting a "-" operator in the expression by "-1*", a "1" being in exponential format; and

(dh) recasting a "division by a constant" in the expression as multiplication by a reciprocal of the constant;

wherein said reducing said strings in accordance with the set of predetermined simplifying rules in step (b) comprises:

(ba) arranging token pairs into subgroups;

(bb) arranging operand tokens in an arranged subgroup in order;

(bc) reducing the ordered operands by consolidating one or more constants and eliminating variables of opposite effect to form reduced subgroups; and

(bd) consolidating one or more multiple instances of similar subgroups, to produce a reduced string.

21. (New) The computer program product of claim 20, wherein said determining the equivalence

of the two algebraic expressions comprises prior to the recasting step (a), in relation to said algebraic expressions, all of the preconditioning sub-steps.

22. (New) The computer program product of claim 20, wherein an algebraic expression whose equivalence is to be determined contains an aliased variable, said determining the equivalence of the two algebraic expressions further comprising the steps of:

arranging an ordered list of aliases of the variable, and substituting a first alias in the ordered list for all instances of the aliased variable in the expression, wherein said compiling comprises said arranging an ordered list of aliases of the variable and said substituting a first alias in the ordered list.

23. (New) The computer program product of claim 20, wherein an algebraic expression whose equivalence is to be determined contains a function, said determining the equivalence of the two algebraic expressions further comprising the steps of:

reducing function arguments using the set of predetermined simplifying rules; and replacing the function by a tagged string, said string designating a function name, parameter types, and arguments, wherein the tag distinguishes the function name from a variable, wherein said compiling comprises said reducing function arguments and said replacing the function by a tagged string.

24. (New) A computer system comprising a processor and a computer readable memory unit coupled to the processor, said memory unit containing instructions that when executed by the

processor implement a method for performing compiler optimisation of source code during compilation of the source code in a computer environment by determining and utilizing the equivalence of two syntactically correct algebraic expressions comprised by the source code, said method comprising compiling the source code into object code, said compiling comprising determining the equivalence of the two algebraic expressions followed by eliminating a redundant algebraic expression of the two algebraic expressions determined to be equivalent, said determining the equivalence of the two algebraic expressions comprising the steps of:

(a) recasting said expressions into a form of one or more token pairs arranged sequentially in a string, each said token pair comprising an operator followed by an operand;

(b) reducing said strings in accordance with a set of predetermined simplifying rules;

(c) comparing the reduced strings by matching, to detect equivalence of the two algebraic expressions;

wherein said determining the equivalence of the two algebraic expressions comprises prior to the recasting step (a), in relation to said algebraic expressions, at least one of the following preconditioning sub-steps:

(da) deleting a space in the expression;

(db) removing a bracket in the expression by expanding a bracketed sub-expression;

(dc) inserting a unitary operator at the start of the expression;

(dd) recasting a power factor, being a variable being raised to a power in the expression, in an alternate form as one of:

(dda) the power factor being expressed as the variable multiplied by itself as many times as the power, if the power is a positive integer;

(ddb) the power factor being expressed as a reciprocal of the variable multiplied by itself as many times as an absolute value of the power, if the power is a negative integer;

(ddc) the power factor being replaced by an appropriate function which can compute the power factor, if the power is not an integer;

(de) recasting a constant in the expression in exponential format;

(df) substituting a "+" operator in the expression by "+1*", a "1" being in exponential format;

(dg) substituting a "-" operator in the expression by "-1*", a "1" being in exponential format; and

(dh) recasting a "division by a constant" in the expression as multiplication by a reciprocal of the constant;

wherein said reducing said strings in accordance with the set of predetermined simplifying rules in step (b) comprises:

(ba) arranging token pairs into subgroups;

(bb) arranging operand tokens in an arranged subgroup in order;

(bc) reducing the ordered operands by consolidating one or more constants and eliminating variables of opposite effect to form reduced subgroups; and

(bd) consolidating one or more multiple instances of similar subgroups, to produce a reduced string.

25. (New) The computer system of claim 24, wherein said determining the equivalence of the two algebraic expressions comprises prior to the recasting step (a), in relation to said algebraic expressions, all of the preconditioning sub-steps.

26. (New) The computer system of claim 24, wherein an algebraic expression whose equivalence is to be determined contains an aliased variable, said determining the equivalence of the two algebraic expressions further comprising the steps of:

arranging an ordered list of aliases of the variable, and substituting a first alias in the ordered list for all instances of the aliased variable in the expression, wherein said compiling comprises said arranging an ordered list of aliases of the variable and said substituting a first alias in the ordered list.

27. (New) The computer system of claim 24, wherein an algebraic expression whose equivalence is to be determined contains a function, said determining the equivalence of the two algebraic expressions further comprising the steps of:

reducing function arguments using the set of predetermined simplifying rules; and replacing the function by a tagged string, said string designating a function name, parameter types, and arguments, wherein the tag distinguishes the function name from a variable, wherein said compiling comprises said reducing function arguments and said replacing the function by a tagged string.